TABLE OF CONTENTS

	Page	
SUMMARY	1	1/A4
INTRODUCTION	1	1/A4
PROBLEM DESCRIPTION	2	1/A5
TECHNICAL APPROACH	3	1/A6
RESULTS AND DISCUSSION	4	1/A7
System Selection	4	1/A7
Mathematical Model Selection	5	1/A8
Array Processor Programming	6	1/A9
Multilooping	8	1/A11
Limitations	11	1/A14
CONCLUSIONS	13	1/B2
APPENDIX A – AP-120B DESCRIPTION	15	1/B4
APPENDIX B – ARGUMENT NORMALIZATION AND TABLE LOOK-UP	17	1/B6
APPENDIX C – ITERATIVE DIVIDE ALGORITHM	19	1/B8
REFERENCES	22	1/B11

NASA1.60:1267

NASA Technical Paper 1267

COMPLETED
ORIGINAL

Application of Special-Purpose Digital Computers to Rotorcraft Real-Time Simulation

D. Brian Mackie and Seth Michelson

JULY 1978

NASA

31

NAS1.60:1267

NASA Technical Paper 1267

Application of Special-Purpose Digital Computers to Rotorcraft Real-Time Simulation

D. Brian Mackie
Ames Research Center
Moffett Field, California

Seth Michelson
Computer Sciences Corporation
Mountain View, California

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Office

1978

APPLICATION OF SPECIAL-PURPOSE DIGITAL COMPUTERS TO ROTORCRAFT REAL-TIME SIMULATION

D. Brian Mackie
Ames Research Center

and

Seth Michelson
Computer Sciences Corporation

SUMMARY

A study was initiated to determine the suitability of using an array processor as a computational element in rotorcraft real-time simulation. It was necessary to determine whether the speed of such a processor would be great enough to accurately simulate complex rotorcraft. Since memory limits on such a computer are quite restrictive, only the rotor portion of the model could be array processor resident. The array processor proved fast enough to execute the rotor code in less than 5 msec. Thus, the course of investigation branched into a study of the validity of a multilooping scheme, in which the rotor would loop over its calculations a number of times while the remainder of the model cycled once on a host computer. To prove that such a method would realistically simulate rotorcraft, a FORTRAN program was constructed to emulate a typical host/array processor computing configuration. The multilooping of an expanded rotor model, which included appropriate kinematic equations, resulted in an accurate and stable simulation.

In the course of the study, many programming and operational difficulties were encountered. All of these were due to a fundamental conflict of concepts between the general-purpose program and the special-purpose computer. Some of these problems were mere inconveniences. However, several severe problems were also encountered during the development of the array processor program. In particular, major manpower effort was required for translation of a FORTRAN model into microcode, and logical debug would further require substantial effort.

INTRODUCTION

When using a rotating blade-element method in modeling rotorcraft, an entirely new and more complex set of aerodynamic considerations arises than those encountered with fixed-wing aircraft. The initial complications arise because the rotor blades induce lift and drag on their surfaces as a function of the radial distance from the hub to a point on the blade. The process is further complicated by the fact that at any given moment, each blade is encountering a unique set of flight conditions. Therefore, each must be considered individually. Finally, high-frequency contents are introduced into the entire aircraft model through rotor rotational moments.

At the Ames Research Center, it has been determined that for a simulation of conventional fixed-wing aircraft to be acceptable to the engineers and research personnel, the integration interval (cycle time) should be less than approximately 50 msec. Otherwise, both the motion and visual systems will generate granular, unrealistic results, and numerical instabilities can be introduced into the dynamical equations. But, because of the inherent complexities involved in rotorcraft simulations, another aspect of the problem had to be considered. To maintain numerical stability and model fidelity for rotating blade-element rotor models the rotor must cycle at approximately 5 msec (see ref. 1). Because Ames simulation facilities have been structured for fixed-wing aircraft, the computational capabilities for handling complex rotorcraft were totally insufficient.

Recognizing the need for rotorcraft simulation, and recognizing the inadequacies of the present simulation environment, this study was undertaken to determine whether an array processor could be used as the dedicated element in a distributed computing system on which the rotor portion of the rotorcraft mathematical model could be run. This examination further resulted in the study of multilooping techniques which were designed for the parallel processing of the aircraft equations with multiple repetitions of the rotor equations. Further, the study was designed to investigate not only the technical feasibility of such an approach, but to define the limitations encountered when using a special-purpose processor to solve a general-purpose problem.

PROBLEM DESCRIPTION

An area of significant interest in real-time simulation at Ames Research Center deals with the simulation of complex rotorcraft. The problems arising from rotorcraft simulations are twofold. First, the large number of equations contained in rotorcraft models requires large simulation cycle times on Ames computers. As noted earlier, previous experience with real-time simulations had indicated that cycle times should be under 50 msec. Due to bandwidth limitations on existing Ames computers, it was not possible to achieve this cycle time for complex rotorcraft models. Second, the frequency content of the rotating blade-element rotor model equations requires that cycle times be of the order of 5 msec. Because the existing simulation laboratory computers could not meet the 50-msec cycle time requirements for this model, these further demands could not be met.

Because of the computer bandwidth limitations, a number of methods have been used in the past to achieve real-time execution of rotorcraft simulations. The most common methods have been to degrade the rotor representation and/or the integration interval. A recent study of the application of these methods to a typical rotating blade element model indicated that the rotor cycle time should be less than 5 msec (see ref. 1). Increasing the integration interval beyond 5 msec resulted in increasingly incorrect static and dynamic vehicle response as a result of larger rotor blade azimuthal advance angle.

Therefore, it became necessary to determine how this particular rotor model could be cycled in less than 5 msec without degrading the rotor representation. Two possible solutions were examined. First, a study was initiated to see if a large, high-speed digital computer could solve this type of problem. It was determined that while such a computer could easily meet speed and programming requirements, the cost in terms of manpower, time, and money warranted an investigation into other possible systems (ref. 2). Concurrently, a second study was undertaken to investigate four alternative systems: (1) low-cost general-purpose digital computers; (2) analog

systems; (3) hybrid systems; and (4) special-purpose digital peripheral computers. Results of this second study indicated that a system composed of a special digital peripheral attached to a general-purpose host mainframe would be the most effective approach (see ref. 3).

Having now established the desired approach, a market survey was undertaken to consider the set of special-purpose peripherals that could be used for the simulation effort. Specifics of each machine were discussed, including manufacturer hardware, software, and support. The results of this survey are contained in reference 4.

The specific problem under consideration in this paper is whether a system consisting of a special-purpose digital computer attached to a host mainframe can accurately simulate complex rotorcraft in real-time. Using a representative computer system and rotorcraft model, results of array processor programming studies and alternative multilooping configurations are presented. Further, limitations of such an approach are assessed.

TECHNICAL APPROACH

The technical approach chosen for this study consisted of the following steps. First, it was necessary to select a representative computer system. Based on studies cited earlier, it was determined that this system should consist of an array processor and an associated host computer. The array processor would nominally contain the rotor model plus any other required computations or portions of the rotorcraft model which could be easily programmed in microcode. The host computer would contain any parts of the rotorcraft model not programmed on the array processor, control array processor execution, and provide required interface to the "outside world."

It was also necessary to select a rotorcraft mathematical model to use in the analysis. The objective was to choose a model that (1) was typical of the type Ames Research Center would be using in the next 5 years, (2) could be programmed or modified in a reasonable time, and (3) would have readily available static and dynamic checks.

The next step in the study consisted of programming the representative rotorcraft mathematical model for the particular computer system chosen. The goal was (1) to acquire actual hands-on experience with array processor programming techniques, and (2) to develop a reasonable estimate of the expected cycle time. It was expected that this phase of the study would indicate any difficulties in this approach from a computer science point of view, and give a general estimate of the potential of such a system.

Finally, it was necessary to investigate multilooping configurations, in which the high-speed array processor would execute all of its code several times during one loop of the slower host computer. This effort was directed toward determining whether a stable and sufficiently accurate real-time solution to the rotorcraft mathematical model could be obtained using a combination of the host computer and multiple loops on the array processor.

RESULTS AND DISCUSSION

System Selection

As stated earlier, it was determined that the most advantageous system considered would be one in which a fast peripheral computer was attached to a general-purpose host mainframe. The actual system configuration chosen for this study consisted of a Floating Point Systems AP-120B attached to an unspecified host mainframe. A short description of the AP-120B can be found in appendix A. Because the AP-120B includes a FORTRAN-based emulator, it was possible to program the rotor for the array processor and to obtain cycle time estimates without obtaining the actual hardware. The multilooping concept of the host/AP-120B system was simulated using a FORTRAN model of the entire rotorcraft. Using time-scaling techniques on the simulation, several multilooping configurations were tested allowing results to be obtained for host computers of several different speeds.

The hardware and software aspect of both the host computer and the array processor are discussed in more detail below.

Host computer– Because the array processor is a peripheral device, it must be physically attached to a host computer. The host computer is a large, general-purpose digital mainframe, which is responsible for controlling execution of the array processor and the transfer of its data. In addition, the host computer must accommodate all real-time operating software and any portions of the rotorcraft model not resident in the array processor.

The present environment at Ames Flight Simulation Laboratory would likely dictate the choice of a host from an already existent set of computers. However, some facilities might require the purchase of a new computer as a host. The specific factors of such a decision are not within the scope of this study but some general statements can be made about the kind of host mainframe that should be used.

Primarily, the host mainframe should have enough programmable memory to completely hold the modified aircraft model and all real-time support software. Furthermore, it should be fast enough to execute them in a reasonable time. Secondarily, the computer should be adaptable to the present operating environment. Specifically, the real-time support code for the rotorcraft model should be of the same form and substance as existing real-time software to maintain facility uniformity. If a new machine is to be obtained for use as the host mainframe, both time and effort would have to be expended in the development of a compatible real-time operating system. A third factor to consider is that the array processor and the host mainframe be hardware compatible. Some array processor manufacturers provide hardware interfaces for a select set of host candidates. However, if the machine chosen to be the host is not a member of that set, extra money and manpower would have to be allocated for the development of such a piece of equipment. Finally, perhaps the most important consideration involved with the host computer is that the data transfer and array processor control software run efficiently and quickly. If not, any savings realized by implementing the rotor on a fast peripheral computer could be lost in the transfer overhead.

Array processor– To pursue this line of investigation, one representative, special-purpose digital computer was chosen on which to implement the rotor model. The choice in this case was

based primarily on the results of the market survey initiated by Ames Research Center and reported in reference 4.

Some general statements can be made about the kind of special-purpose digital computer that might be utilized in a real-time simulation environment. First, it should be large enough to hold the rotor model, all associated library and user-written routines, communications software, plus certain other short routines to be described later. In cases where available program source memory is limited, this consideration is by no means trivial. Second, the computer should be fast enough to execute all of this code in a cycle time short enough to produce an accurate real-time solution to the rotor model. Various special-purpose digital computers have internal clock cycles in the 6 MHz to 40 MHz range. At the lower end of this range, extensive equation sets in the rotor model could cause the cycle time to become too long. Third, as noted in the discussion of the host computer, the array processor and the host must be hardware compatible.

Mathematical Model Selection

Selection of an appropriate rotorcraft mathematical model to use in this study required that a number of factors be taken into consideration. As noted earlier, three criteria which the mathematical model should satisfy were: (1) applicability to future Ames Research Center simulation studies, (2) ease of programming and modification, and (3) availability of checkout data. The rotorcraft mathematical model chosen to perform this study was a preliminary model of the Rotor Systems Research Aircraft (RSRA).

The RSRA is a flying test platform intended for use in the evaluation of advanced rotor and control system concepts. It is a complex rotorcraft which can be configured as a helicopter (main rotor, tail rotor, and upper horizontal stabilizer), a compound rotorcraft (helicopter plus wings, turbo fan engines, and lower horizontal stabilizer), or a fixed-wing aircraft (wings, turbofan engines, upper and lower horizontal stabilizer). The RSRA mathematical model is a total-force, nonlinear, large-angle representation in six rigid-body degrees of freedom. The rotor model is a full rotating blade-element model with representation of rotor blade flapping, lagging, air mass flow, and hub rotational degrees of freedom.

The RSRA model was selected for use in this study for a number of reasons. First, the RSRA is a flying test platform for advanced rotor and control system studies. In preparation for these studies, the simulation model will be utilized extensively over the next several years. Second, real-time simulation studies of the RSRA have been performed in the past at Ames Research Center. Thus, an RSRA simulation program was already in existence and could be easily modified to perform this study. Furthermore, the simulation program could be run on the existent simulation laboratory system, which included facilities for easy output of static and dynamic check data. Finally, a significant amount of previous research had been performed on the RSRA. In particular, studies on rotor model degradation (see ref. 1) and simulations using parallel processors (see ref. 5) provided valuable guidelines in the use of the RSRA mathematical model for this study.

Array Processor Programming

Before starting to microcode, the first step in the implementation of the RSRA simulation model on the array processor was the creation and debugging of a FORTRAN program of the entire model on the host computer. The Ames Research Center simulation laboratories are well equipped with both foreground (real-time or time-scaled) and background (batch mode) programs to aid the simulation engineer in debugging a FORTRAN program. Aiso, as will be discussed further in a later section, the array processor simulator would have required several hours to complete just one pass through the rotor subroutine. Thus, static and dynamic checks required to debug the entire simulation, or even that part to be run on the array processor, would have been unobtainable in a reasonable length of time without a FORTRAN version of the model.

Preparation of the FORTRAN model was done in such a way that the program could be easily adapted to the array-processor system. COMMON blocks were organized such that variables to be transferred between the host and the array processor were contained in separate, contiguous blocks This simplified the transmission of variables immensely. In the subroutine which would be programmed later on the array processor, calls to argument normalization routines for nonlinear function table look-ups were placed, where possible, immediately after the calculation of the independent variable. Reciprocals of constants allowed some divisions to be replaced by multiplication. Sets of equations which could be written in vector notation were grouped together as much as possible. Finally, the subroutine code was carefully optimized to minimize the number of lines, keeping in mind that each FORTRAN line would require an average of seven to eight lines of array-processor microcode to implement.

During the programming of the FORTRAN model it was necessary to determine which groups of calculations could be written in vector (or matrix) notation. This would allow efficient use of the matrix and vector manipulation library routines provided for the array processor. Grouping these equations together allowed sequential calls to vector or matrix routines to be combined into one call which reduced overhead and increased execution speed.

Since the array processor program is written on the microcode level without the aid of a symbol table builder, it was necessary to allocate array processor memory prior to writing the array processor code. Accessing sequential memory locations is much easier and faster than accessing random memory locations. If high-speed data memory is being used, only variables on different pages of the interleaved memory may be accessed in sequential clock cycles. Thus, it was essential that the address of each variable in memory be allocated before coding began.

For the RSRA real-time simulation program, memory was allocated so that variables were grouped into the following major categories:

1. Vectors: Typically, vectors had only three components corresponding to the three rotational or translational degrees of freedom.

2. Matrices: Corresponding to the length of vectors, matrices were typically 3×3 square matrices.

3. Arrays: Variables that were dimensioned as arrays in the FORTRAN program could be most easily accessed if placed in a separate location in memory.

4. Data tables: For this application, table data were assumed to reside in the Main Data Memory. If read/write table memory were purchased, table data could be placed in table memory. This would require some slight modification to the present nonlinear function generation routines, and would perhaps result in a slight decrease in execution time.

5. Data table headers: Each data table required a header for use by the nonlinear function generation routines. The header was actually composed of one 7-word vector for each independent variable associated with the data table. Argument normalization and table look-up procedures on the array processor are discussed in appendix B.

6. Divide space: The iterative Newton-Raphson divide algorithm implemented for this study typically requires locations for the storage of the inverse, if it is not explicitly called for in the FORTRAN program; this divide algorithm is discussed later in the report and in appendix C.

7. Constants: Literal values (e.g., constants in equations) must be placed in memory at a specific location, and accessed by address. Thus, the addressing mode typically referred to as "immediate addressing" is not provided. As with data tables, if table memory is purchased, these constants could be placed in table memory, resulting in a slight decrease in execution time (167 nsec decrease per access).

8. Real variables: For this study, real variables were placed in memory in alphabetical order without concern for order of access. For a large simulation program, in which variables are randomly accessed at a number of points throughout the program, attempting to arrange variables according to order of access is probably not practical.

9. Integer variables: Because integer variables are treated differently from real variables in the program, it was easier to place them in a separate area in memory.

The above memory organization was by no means optimized to place sequentially accessed variables in alternate memory pages, nor was it the only one possible. However, based on the authors' experience it was a workable organization which allowed the program to be written in a straightforward manner.

Once the FORTRAN program had been completely debugged, the equations of vector form grouped together, and the array processor memory allocated, the writing of the array processor program could begin. To facilitate debugging of the array processor code, the following procedure was followed. Typically, one FORTRAN line was translated into array processor microcode before proceeding to the next line. This is not to imply that look-ahead was ignored. Significant savings were gained by overlapping some FORTRAN calculations to fill a pipeline or by saving intermediate results in scratch pad registers for future reference. As each FORTRAN line was encountered, it was written as a comment line above the corresponding section of microcode. Thus, at debug time, it would be possible to verify any questionable section of array processor microcode by executing the FORTRAN code to the corresponding line and comparing the variable(s) in question.

To take advantage of the array processor's speed, it was necessary to determine at each line of code which instructions (e.g., add, multiply, memory fetch) could be performed in parallel and which operations could be easily pipelined. Also, because of memory access timing it was neccessary to anticipate which variables would be required and fetch them at least three clock cycles in

advance of the time they would be needed. These three considerations (paralleling, pipelining, and memory access) combined to greatly complicate the programming of the array processor. Without careful accounting for each variable throughout the program, the difficulties of programming could have become overwhelming.

On the AP-120B emulator, any syntax errors in the array-processor code were flagged by the assembler at assembly time. Once these errors had been corrected, the modules created by the assembler were linked for debugging or for writing into the array processor for execution.

Timing results for the array processor code could have been obtained in two ways. Using the interactive debugger, it would have been possible to execute the program and obtain timing results directly using the breakpoint capability. However, as will be discussed later, execution of the entire program with the debugger would have been an unacceptably time-consuming task. Thus, in this case, timing results were obtained by taking into account all executable lines of code and multiplying by 167 nsec/line. Memory requirements were obtained directly from the load map created by the loader.

Memory and timing results for the blade-element rotor model programmed for this study are as follows: The FORTRAN model consisted of 276 lines of code, requiring 1504 procedure words and 3898 data words of memory on an XDS Sigma 8 computer. Under the extant operating system, the rotor code will execute in approximately 37 msec on the Sigma 8. The equivalent program implemented on the FPS AP-120B (based on results using the emulator) would require 2054 words of Program Source memory (corresponding to the required procedure words above) and 3116 words of Main Data memory (corresponding to the required data words above). Timing results indicate that this code will execute in approximately 4.7 msec on the AP-120B.

Because the array processor code was not debugged, as discussed later, the results for the AP-120B are not absolute. However, even allowing as much as a 10% error in the coding (which would correspond to adding 200 lines of array processor code), the array processor exhibits better than a 7-to-1 speed advantage over the Sigma 8 for this mix of instructions. Programming complexity of the array processor is much greater, though, with each FORTRAN line requiring an average of 7.4 lines of parallel microcode to implement. Note also that the 2054 words of Program Source memory, which include all library and user subroutines required in the array processor, are slightly more than half the available maximum of 4096 words of AP-120B program memory.

Multilooping

Initially it had been hoped that the majority of the rotorcraft model could be array processor resident. However, once the rotor portion had been microcoded, it became apparent that the remainder of the rotorcraft model could not be included on the array processor, because the rotor code required over half the available program source memory on the array processor. Thus, the course of investigation branched into a study of the validity of a multilooping scheme.

It was not clear at the outset whether separating the rotor from the rest of the aircraft and calculating the rotor equations at a smaller cycle time in parallel (i.e., multilooping) would result in a reasonable simulation of the entire aircraft. Since multilooping is the basis for the entire array processor approach, a FORTRAN simulation was designed from an existent simulation model

of the RSRA. The simulation consisted of a variable repetition loop to enclose the "array processor code," a communications routine to simulate a parallel processing communications link, and a pair of variable cycle times to simulate real-time multilooping. Various static and dynamic checks were then performed on this "host/array processor" system.

The emulation, with this particular loop structure, allowed the option of choosing the ratio of the number of times the rotor would cycle to each cycle of the aircraft code. Since the rotor calculations had been estimated to require approximately 5 msec on the array processor, the time-scaled aircraft frame time was $5N$ msec, where N is the number of rotor cycles per aircraft cycle. Many different values of N were tested so that the results could apply to a variety of hardware configurations. Thus, static and dynamic checks for loop ratios from 1-to-1 (aircraft frame time of 5 msec) to 12-to-1 (aircraft frame time of 60 msec) were carried out for several systems and aircraft configurations. The single computer version of the model (i.e., no multilooping) was run at 5 msec and used as a control case.

The most significant effects on the aircraft as far as system stability is concerned, will manifest themselves in the rotational degrees of freedom, due to their high-frequency components. Other studies indicate (see ref. 1) that because of the relatively small inertial values in the roll axis, the model is most sensitive in this degree of freedom. Hence, the aircraft model is more likely to become unstable in this critical area than in the pitch or yaw axes. Therefore, inputs that excite the roll degree of freedom (lateral cyclic pulses) were used as a worst case approach to testing the model stability.

The total vehicle dynamic response tests were initiated by starting the vehicle at a trim condition determined by the simulation model trim routine. At 1 sec into the flight, a 5% lateral cyclic pulse was applied for 1 sec. Appropriate variables representing vehicle response were plotted by strip-chart recorders for analysis. Of these variables, body roll acceleration (PBD), roll rate (PB), roll angle (PHI), and rotor blade flapping angle (BR) are presented in figures 1–4.

Figure 1 displays the total vehicle dynamic response with only the rotor on the "array processor" and the remainder of the aircraft model on the "host computer." The responses at loop ratios of 1-to-1 and 2-to-1 are indistinguishable from the single computer response. However, at the 4-to-1 loop ratio, there is a slight high-frequency oscillation in body roll acceleration (PBD). At a loop ratio of 6-to-1, the oscillation in PBD is very pronounced and divergent with effects of the oscillation appearing in body roll rate (PB). At a loop ratio of 8-to-1, there are divergent oscillations in PBD, PB, body roll angle (PHI) and rotor blade flapping angle (BR). The 8-to-1 loop ratio represents an aircraft cycle time of 40 msec, which is less than that expected on the Ames host mainframe. Thus, the configuration in which just the rotor is array processor resident does not represent a feasible method of simulating complex rotorcraft for cases where the host computer cycle time is greater than 20 msec.

In this configuration, the rotor is separated from the body rotational accelerations. Thus, dynamic interaction between the rotor and the rotational accelerations can occur only at the end of every Nth loop. The magnitude of instability increases as the loop ratio N increases. Because of earlier dual processor studies, an alternative configuration is known to reduce instabilities for the equivalent of a 1-to-1 loop ratio at 46 msec (see ref. 5). In this configuration, the body rotational equations are included in the rotor loop, thus allowing higher frequency interactions between the

rotor and the body rotational degrees of freedom. Extending this concept into a multilooping scheme yields the following results.

Figure 2 represents the total vehicle dynamic response with the rotor and the calculation of rotational accelerations and rates on the "array processor" and the remainder of the aircraft model on the "host computer." Again, the responses at loop ratios of 1-to-1 and 2-to-1 are indistinguishable from the single computer response. At the 4-to-1 loop ratio there is a slight low-frequency oscillation in PBD. However, unlike the results from figure 1, the responses at 6-to-1 and 8-to-1 loop ratios are not noticeably different from the response at 4-to-1. Even at loop ratios of 10-to-1 and 12-to-1, the dynamic response is stable and accurate. It should be noted that the rotor blade flapping angle (BR) is not as granular as it appears in figure 2. Due to software limitations and model simulation procedures, the digital-to-analog signals cannot be output every rotor cycle, but only once every aircraft cycle. Results of this experiment indicate that a configuration in which the rotor model plus the computation of rotational accelerations and rates are array processor resident can provide a sufficiently accurate simulation of compound rotorcraft for cases where the host computer cycle time is as great as 60 msec.

Figure 3 shows the total vehicle dynamic response for an array processor cycle time of 6 msec. This case accounts for the effect of increased rotor model complexity. It is possible that a more complex rotor model might be required in other rotorcraft simulations, which could result in dramatic changes to the stability of the system. A cycle time of 6 msec has been chosen because increasing the array processor cycle time from 5 msec to 6 msec would require the addition of approximately 6000 executable instructions. Because program source memory limitations on the array processor would likely preclude the addition of such a large number of instructions, it is not necessary to consider cycle times greater than 6 msec. As in the second experiment, the calculation of body rotational accelerations and rates is array processor resident. Because the dynamic response at 1-to-1 and 2-to-1 loop ratios is indistinguishable from the single computer response, plots of those cases have been omitted. Note that again in the response at the 4-to-1 loop ratio there is a small, low-frequency oscillation in PBD, and the responses at 6-to-1 and 8-to-1 loop ratios are stable and accurate. The response at the 10-to-1 loop ratio, which represents a host computer cycle time of 60 msec, indicates an accurate dynamic response. Results of this experiment indicate that increased rotor model complexity will not adversely affect the dynamic response of the compound rotorcraft simulation.

Figure 4 shows total vehicle dynamic response with the rotorcraft configured as a helicopter rather than as a compound rotorcraft. The computations of body rotational accelerations and rates is again array processor resident. The dynamic response at 1-to-1 and 2-to-1 loop ratios is very similar to the single computer response. The response at the 4-to-1 loop ratio exhibits a slight low-frequency oscillation in PBD. However, at loop ratios of 6-to-1 and 8-to-1, this oscillation is attenuated and the dynamic response is stable and accurate. This experiment indicates that the multilooping scheme can provide a sufficiently accurate simulation of a helicopter as well as a compound rotorcraft.

Further dynamic checks have been performed on both helicopter and compound rotorcraft configurations over a range of flight speeds. These dynamic checks have included lateral cyclic pulses, as in the above experiments, collective pulses, and longitudinal cyclic pulses. Results of these dynamic checks, which are not presented in figures here, indicate that the multilooping scheme in which the computations of rotational accelerations and rates are array processor resident can

provide a sufficiently stable and accurate real-time simulation of rotorcraft, both compound and helicopter.

Limitations

When one attempts to simulate a rotorcraft model in real-time on any computer system, certain limitations are to be expected. Most of these limitations are overcome by reducing the complexity of the computer model, such that approximate accuracy and fidelity are acceptable trade-offs for time critical results. However, when the computer system has been specifically designed to handle a different, special class of problem, an additional set of limitations is encountered that cannot be overcome by reducing the model complexity. Some of these difficulties arise due to the specific machine chosen for the task. But the overwhelming majority of them are the result of conflicts encountered when trying to apply a special-purpose computer to an application for which it was not designed.

While using the array processor, some very serious problems were encountered due, primarily, to this general conflict in design philosophy. It is important to remember that the array processor was designed for use in small problems with large amounts of vector data. Examples of such applications are signal processing and seismic analysis. The rotor model, on the other hand, is a large set of differential equations with many scalar variables and a small set of three-component vectors.

Array processors were designed to be used in applications in which the typical program is rather small, although the data set may be immense. Therefore, the basic hardware configurations offered on the array processors feature a very small program source memory space. At first it was hoped that an entire simulation could be implemented on the peripheral computer with only real-time operating software remaining on the host. But because the AP-120B has a source program limit of only 4096 words of memory, the only alternative for this study was the implementation of just the rotor model on the array processor. This does not present a serious problem, if the type of model will lend itself to a multilooping scheme. In the case of the RSRA, with certain modifications, this was true. However, the small memory size is a severe limitation to the further addition of control systems and/or the complication of the model.

Array processors are designed for applications in which floating point division is not generally required. On the AP-120B no floating divide hardware is built into the unit, but a slow software divide is provided instead. However, the rotor model has 84 floating divisions. The slowness of the provided software necessitates the adoption of a simpler, though less accurate, algorithm to do software divides. This requires the design, implementation, and debugging of another piece of software. While there is a possibility that table memory could be purchased to hold the divisor and dividend, that would only speed up the algorithm by one instruction per division. The specifics of the divide algorithm are discussed in appendix C.

Because the design philosophy is at odds with this application it is not unreasonable to conclude that the programming philosophy should also be at odds. The array processor was designed such that maximum efficiency could be achieved by writing short, tight loops in which all the independent hardware units were being used simultaneously. The efficiency is further increased if the data set can be considered a large vector which is to be processed by these loops. The rotor model used in this study was not formulated in this manner at all. There were few vectors present,

and those that did exist were only three elements long. The equations were long and complicated, and no short loops could be made from the model. Some of the calculations could be overlapped so that the hardware components could be run in parallel, but this situation was not universal.

The model had some equations that could be written in vector notation, thus making use of matrix-vector algorithms. However, this was not true for the vast majority of the model. The scalar calculations had to be done in specific order and, in many cases, the result of one step was prerequisite to calculations in the next. While some savings were gained by storing results in a fast scratch memory buffer, the theoretical maximum efficiency was unachievable.

As discussed earlier, it was necessary to allocate array-processor memory prior to coding. This memory plan, with conceptual vectors and matrices included, then lent itself to pipelining of memory locations and the processing of data. Typically, a software library of vector functions is provided by array processor manufacturers. However, because the vectors encountered in this model were so short, the overhead entailed in library utilization made this practice counter-productive in a real-time environment. This situation is not a function of any particular machine or software but rather, the result of the basic conceptual conflicts encountered in trying to apply a special-purpose processor to a class of problems for which it was not designed.

Another problem that became apparent was the large manpower effort required to translate a FORTRAN program to microcode. The efficiency and speed of an array processor can only be achieved by coding every clock cycle, causing the translation of any appreciable FORTRAN program to require an incredible manpower effort. As an example, on the given RSRA model, an average simulation engineer can completely code the rotor in FORTRAN in about 2 weeks. (Note, this is the required time for coding only and does not take into consideration study and preparation time needed to understand the mathematical model. However, that adidtional time would be overhead in considering coding effort and is irrelevant to this point.) The actual translation time needed to microcode the base FORTRAN model was 3 man-months, involving experienced personnel. The work, in addition to being very slow, was very difficult to master. Many idiosyncrasies of microcode significantly complicated the process of direct translation.

Problems also arise with respect to system software due to this conceptual conflict. While the exact problems are a function of the particular software, the general class of problems encountered is a consequence of the philosophy. Unrealistic limits on symbols and labels result because of the assumption that the machine will only be used for small programs. On the AP-120B in particular, no limit checks were done on the arrays in the asembler, which handled the user symbols, thus giving rise to spurious and misleading error messages. Further assumptions were made in the emulator with respect to the particular hardware configuration (memory size and speed) of the array processor being simulated. These required a great deal of work to correct in both the linker and the emulator. The additional time required to first define, and eventually fix, these problems was very costly to the program development effort. These problems are the direct consequences of the assumptions made concerning applications of the machine.

The problems with software increased during debug time. At Ames flight simulation facilities, real-time foreground work is being done in parallel with nontime-critical software development in background. Because the simulator was designed to mimic a small array processor configuration, no initial problems arose with its use. However, once the rotor code was translated only a large array processor would accommodate it. The simulator program grew by a factor of about 3.5, from 15.6K

to 56.6K, putting the background software development process in conflict with the real-time user. The majority of the time slice allocated to the background user was spent in paging the program into and out of working core. The result was that the emulation of 38.5 μsec of code required 40 clock minutes. At this rate, it would have taken approximately 82 hr to complete one emulator pass through the rotor code in background on the Ames simulation computer.

The consequences of such a situation would mean that program debugging would require foreground status and thus, scheduling and system software support, or, alternatively, a dedicated computer, to avoid swapping and paging. Besides these problems, the debugging of the microcode is a very slow and arduous process. A period of 6–8 weeks for debugging the array processor rotor model would be a reasonable estimate. Therefore, any simulation would require dedicated foreground computer time for at least 2 months prior to its scheduled execution date.

CONCLUSIONS

This study was designed to investigate the feasibility of employing a relatively fast peripheral computer on which the rotor portion of a real-time rotorcraft model could be executed. The overall investigation branched into three subareas, each being fundamental to the feasibility of this approach. First, it was necessary to determine array processor speed factors. Second, the numerical feasibility of multilooping as a simulation method had to be tested. And third, the practical limitations encountered due to incorporation of a special-purpose digital processor into the general simulation environment had to be defined and evaluated.

The key points of interest seemed to separate into two natural categories. The first is the feasibility of the scheme with respect to the simulation environment, and the second is the feasibility of all the computer science aspects of the method. While in an actual operating environment the ultimate effects are totally interrelated, a study such as this, by its very structure, distinguishes between them.

The most important point concerning the feasibility of simulating rotorcraft in this manner is that, because the array processor can cycle the rotor in under 5 msec (a factor of 7 speedup over the present Ames' simulation computer) the multiloop method, with suitable rotor support code, successfully achieves the degree of accuracy and stability needed to make this a workable approach. While it was shown that the software configuration which separates only the rotor from the remaining aircraft grew unstable at such small loop ratios as 6-to-1, the addition of body rotational acceleration and rate equations to the high frequency component returned the system to stability. Not only did it become stable, but the system exhibited remarkably accurate results at such high loop ratios as 12-to-1. As a simulation tool, the multilooping method for rotorcraft is quite valuable.

However, new aspects of the problem present themselves when the issue of practicality is considered. There are many problems encountered in the area of computer science that are due entirely to the conceptual conflicts encountered between the special-purpose computer and the general-purpose problem. Some of these limitations can be overcome with a minimum of difficulty. They include the small number of symbols allowed by the initial emulator and the lack of a hardware divide. Though they result in some inconvenience, there are methods for circumventing

them. However, the software development aspect of the problem is quite a different matter. The need for an original FORTRAN model as a basis for translation to microcode can require an extra development effort, separate from the array processor coding. The actual translation process is a very slow and arduous one. Major manpower efforts are required of skilled programmers to translate the initial FORTRAN model into microcode. Not only is the effort considerably greater than that usually required in writing a FORTRAN model for the given specifications, but the degree of talent available to do the work could very well be the determining factor in the success of such an approach. Furthermore, once the model is microcoded, significant problems may be encountered during logical debug. Even if the emulator/debugger process is given foreground status, and it is possible that it may require total stand-alone capability, the debugging of such a lower level language will require an extensive amount of time since it is known to be much more difficult to debug than FORTRAN.

In conclusion, a special-purpose digital computer can be utilized as a relatively low-hardware-cost, high-speed computational element in rotorcraft real-time simulation. Given a rotor model that requires a short cycle time, an array processor can provide the necessary computing bandwidth for dynamic stability. With the addition of appropriate kinematic equations to the array processor, the entire rotorcraft model can be accurately simulated using a multilooping scheme with a slower host computer. Given the limitations discussed above, it should be realized that only a limited class of rotorcraft models can be simulated, that the development of the simulation would require major manpower efforts, and that changes to the simulation model could require extensive implementation time.

Ames Research Center
National Aeronautics and Space Administration
Moffett Field, California 94035, February 22, 1978

APPENDIX A

AP-120B DESCRIPTION

Hardware

The specific array processor chosen to perform this study is the Floating Point Systems AP-120B. The AP-120B is a high-speed (167-nsec cycle time) peripheral floating point arithmetic array processor, intended to operate in parallel with a host computer. Figure 5 is a general block diagram of the system elements and arithmetic paths, including a possible interconnection to the host computer.

The system elements are interconnected with multiple, parallel arithmetic paths so that data transfers may occur simultaneously. All internal floating point data paths are 38-bits wide (10-bit biased exponent plus 28-bit mantissa). The interface unit is available for specific host computers and allows either I/O or DMA channels to be utilized for data transfer. Format conversion during data transfers is performed automatically in the interface unit.

Operation of the unit is controlled by the execution of 64-bit instruction words residing in Program Source Memory. Program Source Memory is available in 256-word modules up to a maximum of 4K words. Integer arithmetic and control functions, such as address indexing and loop counting are provided by the S-Pad unit. The S-Pad consists of sixteen 16-bit registers and an integer arithmetic logic unit (ALU).

Main Data Memory is the primary data storage unit for the AP-120B. It is available with 38-bit words, in 8K modules, up to a maximum of 1 million words. Memory operations may be performed every 167 nsec on alternate pages of interleaved memory. However, data "read" from memory are not available in the memory data register until 500 nsec (3 clock cycles) after the read operation is executed.

Table Memory is used to store constants and table data for look-up. It is available in 38-bit word, 512-word modules up to a 65K word maximum. Table memory "reads" may be performed every 167 nsec, and data are available 333 nsec after the read operation. The Data Pad Unit consists of two blocks, each with 32 floating point accumulators. One register may be read and one register may be written in each block in one instruction cycle. The registers are used primarily to store intermediate results for immediate access.

The Floating Point Adder performs addition, subtraction, floating logical, and other floating point operations on the contents of the adder input registers, A1 and A2. The operations are performed in two stages, each requiring 167 nsec. Since the two stages are independent, a new operation may be initiated every machine cycle, with the results being available 333 nsec later. A function such as this, in which the operation may be initiated every clock cycle, but goes through two or more stages before completion is known as "pipelining."

The Floating Multiplier computes the product of the contents of the two multiplier input registers. The operation is performed in three stages, each requiring 167 nsec. Since the three stages

are independent, a new multiply may be initiated every machine cycle, with the results being available 500 nsec later. Floating multiplies and floating adds may be performed simultaneously.

A more complete description of the hardware elements of the AP-120B may be obtained from reference 6.

Software

Software packages available with the AP-120B may be categorized as follows:

1. Executive Routines
2. Mathematical Library
3. Program Development Package
4. Testing Programs

The AP executive program accommodates communication of the host with the AP-120B via a series of FORTRAN or machine language subroutine calls. The executive driver routine interprets the particular user subroutine call and directs the AP-120B to perform the desired task.

The AP Math Library includes 70 subroutines written in AP-120B assembly language and callable either from the host computer, using APEX, or the AP user program. Most of the subroutines are intended primarily for signal processing applications and are not useful for simulation applications. However, the library does include computations of trigonometric functions and vector manipulations that are applicable to this program.

The Program Development Package is composed of four FORTRAN programs which are compiled on the host computer. These four programs are the assembler, linker, debugger, and simulator. The assembler provides a two-pass assembly of user symbolic microcode into an object module, and generates error diagnostics if necessary. The linker links and relocates APAL object modules together into a single executable load module. The debugger is an interactive debugging program which runs on the host computer. It allows the user to set breakpoints, examine and change memory and register contents, and execute selected segments of the program. The simulator is called by APDBUG. These two programs provide an emulation of the AP-120B. All timing characteristics are evaluated, and floating point arithmetic is simulated. Thus, a user can write, assemble, link, run, and debug a new AP program totally off-line without interfering with AP operations.

The testing program consists of a collection of interactive diagnostic tests and verification programs which aid in hardware maintenance. These include tests of interfaces, registers, arithmetic units, and entire system operations.

APPENDIX B

ARGUMENT NORMALIZATION AND TABLE LOOK-UP

This appendix contains a short description of the algorithms used in the nonlinear function generation routines on the FPS AP-120B. Timing results are given for argument normalization and for one- and two-dimensional table look-ups.

Function tables have long been used in real-time aircraft simulation. Because some variables result from empirical testing or statistical analysis, they defy adequate mathematical formulation. Therefore, the large data tables and the concomitant software needed to process them, have to be developed as reasonable alternatives. Among the software needed by simulation engineers are programs for argument normalization and table look-up. Since the tables consist of argument breakpoints and associated function data values which are discrete entities, the arguments calculated in the program must be normalized. In this process, it is determined between which breakpoint table entries the argument falls, and the ratio of the difference between them. The table look-up routine uses this information to interpolate a value from the given data set.

When using breakpoints with equally spaced intervals, the normalization algorithm is rather straightforward. Assume a set of breakpoints $x_0, x_1, \ldots, x_n$ such that $x_{k+1} - x_k = d$, for $k = 0, \ldots, n-1$. Prior to execution time, it is, therefore, possible to determine the inverse of d. Given an argument x, the normalization algorithm proceeds as follows. The argument is initially checked for size. If $x \leqslant x_0$ [or $x \geqslant x_n$] the first (or last) breakpoint is assigned. Otherwise, x_0 is subtracted from x and the result is multiplied by the inverse of d. The integer part of the result is the breakpoint index, i, and the fractional part is the interpolatory ratio, r.

The look-up process is just as straightforward. It is merely a linear interpolation algorithm in one, two, or more dimensions. In one dimension, the equation for interpolation is:

$$f(x) = f(x_i) + [f(x_{i+1}) - f(x_i)] \cdot r$$

In two dimensions, the look-up process proceeds as if two separate one-dimensional look-ups were used to determine two function values for a third one-dimensional look-up. For example, we are given a function of the form $f(x,y)$ with x varying faster than y in the table. Thus, for each y breakpoint, an array of x breakpoints exists. For simplicity, assume that the normalization routine has determined that for a specific argument (x,y) the breakpoint index in each dimension is 1, with associated interpolatory ratios r_x and r_y. The first two interpolations are:

$$f(x,y_1) = f(x_1,y_1) + [f(x_2,y_1) - f(x_1,y_1)] \cdot r_x$$

$$f(x,y_2) = f(x_1,y_2) + [f(x_2,y_2) - f(x_1,y_2)] \cdot r_x$$

These interpolations yield the two points indicated by ▲ in figure 6. Using these two points for a linear interpolation yields:

$$f(x,y) = f(x,y_1) + [f(x,y_2) - f(x,y_1)] \cdot r_y$$

This final value for $f(x,y)$ is indicated by the asterisk in figure 6.

During coding for the array processor, much of these processes are overlapped to take advantage of the pipelining feature. However, some modification of the present Ames algorithm was made as to the manner in which the breakpoint information is communicated from the normalization routine to the look-up programs. While the processes on the array processor and general-purpose computers are analogous, more information, such as the number of x breakpoints per y, and the reciprocal of the interval size, is needed on the array processor. The data table header is also stored in its own memory locations rather than as part of the function table.

The programs were designed, written, and tested on the array processor emulator. The time required to normalize an argument and do a look-up in both one and two dimensions is a function of the arguments being normalized. If the argument is less than the lower limit of the breakpoint table, normalization requires 2 μsec. If it is greater than the upper limit of the breakpoint table, the process requires 2.833 μsec. If it lies between the limits, normalization requires 3.833 μsec. Look-up speed, on the other hand, is independent of the arguments used. One-dimensional look-up requires 3.167 μsec, and two-dimensional look-ups require 6.66 μsec. This represents a factor of 7 speed up, in the worst case, over the XDS SIGMA 8.

APPENDIX C

ITERATIVE DIVIDE ALGORITHM

The Newton Method

The application of iterative algorithms to the solution of systems of equations is an area of extensive study. The Newton method (or Newton-Raphson's method) is one such algorithm which proves to be very useful in digital solutions to certain functions.

Since this algorithm has been so widely studied (see refs. 7 and 8) a much simplified discussion of the method will suffice here. The Newton method is defined as follows: Given: $f(x) = 0$, $f(x)$ is continuously differentiable with first derivative $f'(x)$, and a point x_0 (first estimate of solution).

Compute: for $n = 0, 1, 2, \ldots$

$$x_{n+1} = x_n + \frac{-f(x_n)}{f'(x_n)} \tag{C1}$$

The iterations are concluded when $|x_{n+1} - x_n|$ has become less than the largest error one is willing to permit in the root. It can be shown that this method is quadratically convergent or that it is a second-order method (see ref. 7).

Application to Division

The Newton method can be applied to the solution of the division problem as follows. First, define:

$$f(x) \triangleq \alpha - \frac{1}{x} \tag{C2}$$

Setting $f(x) = 0$, x is the inverse of α. Note that $f(x)$ is not continuously differentiable about $x = 0$. Because the inverse of x is not defined about $x = 0$, we will not be concerned with that case. Differentiating $f(x)$ from equation (C2):

$$f'(x) = \frac{1}{x^2} \tag{C3}$$

Now, substituting $f(x_n)$ and $f'(x_n)$, equations (C2) and (C3), into the iteration formula (C1), we obtain:

$$x_{n+1} = x_n[2 - \alpha x_n] \tag{C4}$$

Thus, an iterative solution to a quotient can be obtained by performing two multiplies and a subtract.

Application to Real-Time Simulation

For a real-time simulation on a special-purpose digital computer, it is possible to apply this algorithm as follows. First, the usual procedure in a real-time simulation is to define several "modes" of operation, two of which are I.C. (Initial Conditions – sometimes called "reset") and operate. In I.C., certain initial values on variables, integrators, filter, etc., are computed or input and the computer execution is not time-dependent. In operate mode all integrators, filters, and differential equations are executed with the requirement that the computer execution be completed within the specified integration interval. For the Newton method, therefore, since I.C. mode is time-independent, it is possible to compute the true reciprocal in I.C. Thus, at the moment the simulation goes into operate mode, a perfect or near-perfect first estimate of the solution is available.

It is necessary to modify the algorithm in equation (C4) slightly because we are trying to determine the inverse in real time of a dynamically varying quantity. Thus, equation (C4) becomes

$$x_{n+1}(t) = x_n(t)\,[2 - \alpha(t)x_n(t)] \qquad n = 0, 1, 2, \ldots \tag{C5}$$

Once the simulation is in operate mode at a given cycle time (integration interval) of Δt, assuming the Δt is sufficiently small and $\alpha(t)$ is smooth, the first estimate for the solution in a given cycle can be obtained from the solution in the preceeding cycle. Thus:

$$x_0(t) = x(t - \Delta t) \tag{C6}$$

Since this method is being implemented to provide a fast solution to the inverse of α, it will often be implemented using only one iteration per cycle. Substituting equation (C6) into equation (C5) yields, for one iteration per cycle:

$$x(t) = x(t - \Delta t)[2 - \alpha(t)x(t - \Delta t)] \tag{C7}$$

Experimental Results

The single-iteration, modified Newton-Raphson divide algorithm given by equation (C7) can be tested using a simple program on a digital computer. Because this algorithm is intended to be implemented on a blade-element rotor model, tests can be performed using a simplified version of the model. Two inverses that are required for the model are $1/\cos\beta$ and $1/V_T$, where β is the blade flapping angle and V_T is the blade element tangential velocity with respect to the wind.

The following simplifying assumptions can be made:

1. Flapping, β, is sinusoidal with maximum excursions of $\pm 15°$.

2. Computer cycle time, Δt, is 5 msec.

3. Tangential velocity, V_T, is determined by rotor angular rate, Ω, and forward velocity, V, of the aircraft. For this example, V_T will be determined at the blade tip.

4. Rotor radius, R, is 9.75 m (32 ft) and rotor angular rate, Ω, is 21.29 rad/sec.

Based on these simplifying assumptions, the equations for $\cos\beta$ and V_T are as follows:

$$\cos\beta = \cos[15\cos\Psi] \tag{C8}$$

$$V_T = \Omega R + V\sin\Psi \tag{C9}$$

where blade azimuth angle, Ψ, is:

$$\Psi = \Omega\Delta t \cdot n\,, \quad n = 1, 2, \ldots$$

Using equations (C8) and (C9), a program can be written to compute the exact inverse of $\cos\beta$ and V_T, compute a single iteration inverse using equation (C7), and determine the percent error between the exact and iterative inverses. Results of this study indicate that the percent error is strongly dependent on the magnitude of the change in the variable from one cycle to the next.

In summary, the results are as follows: Over one complete revolution of the rotor, $\cos\beta$ ranges between 0.966 and 1.0. The maximum percent error for the inverse of $\cos\beta$ is 0.0015% using the single-iteration inverse.

Figure 7 is a plot of V_T and percent error in the single iteration computation of $1/V_T$ as a function of blade azimuth angle Ψ for the 113 m/sec (220 knot) case. At this airspeed, the value of V_T ranges between 95 m/sec (310 ft/sec) and 321 m/sec (1052 ft/sec). Maximum percent error for the single-iteration inverse is 0.5103%. Note from figure 7 that whereas V_T is sinusoidal, the percent error for the single-iteration inverse is cyclic but not sinusoidal. As expected, the percent error is a minimum where V_T is most slowly varying from one cycle to the next (or where the slope of V_T is closest to zero).

For the 62 m/sec (120 knot) case, V_T ranges between 146 m/sec (478 ft/sec) and 269 m/sec (884 ft/sec). Maximum percent error for the single-iteration inverse is 0.1134% for this case. Comparing these results with the previous case, it is apparent that the maximum percent error is strongly dependent on the magnitude of the change in the variable.

Depending on required accuracy, these errors may or may not be acceptable. If there is a requirement for greater accuracy, equation (C5) can be implemented to perform more than one iteration per cycle and increase accuracy to the desired tolerance.

REFERENCES

1. Houck, Jacob A.; and Bowles, Roland L.: Effects of Rotor Model Degradation on the Accuracy of Rotorcraft Real Time Simulation. NASA TN D-8378, 1976.

2. Michelson, S. G.; Alderete, T. S.; and Rieman, F. C.: A Study of Using the CDC 7600 in a Distributed Computing Network for Real-Time Flight Simulation. Computer Sciences Corporation, CSC-PR 6-76, Nov. 1976.

3. Howe, R. M.; and Fogarty, L. E.: Computer Considerations for Real Time Simulation of a Generalized Rotor Model. NASA CR-2877, 1977.

4. Bekey, George A.; and Karplus, Walter J.: Computers for Real Time Flight Simulation: A Market Survey. NASA CR-2885, 1977.

5. Mackie, D. Brian; and Alderete, Thomas S.: A Real-Time, Dual Processor Simulation of the Rotor Systems Research Aircraft. NASA TN D-8328, 1977.

6. Processor Handbook, Rev 02, Form No. 7259. Floating Point Systems, Inc. Portland, Oregon, 1976.

7. Dahlquist, Germund; and Bjork, Åke: Numerical Methods. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1974.

8. Conte, S. D.; and deBoor, Carl: Elementary Numerical Analysis: An Algorithmic Approach. McGraw-Hill Book Company, New York, 1972.

THIS PAGE LEFT INTENTIONALLY BLANK

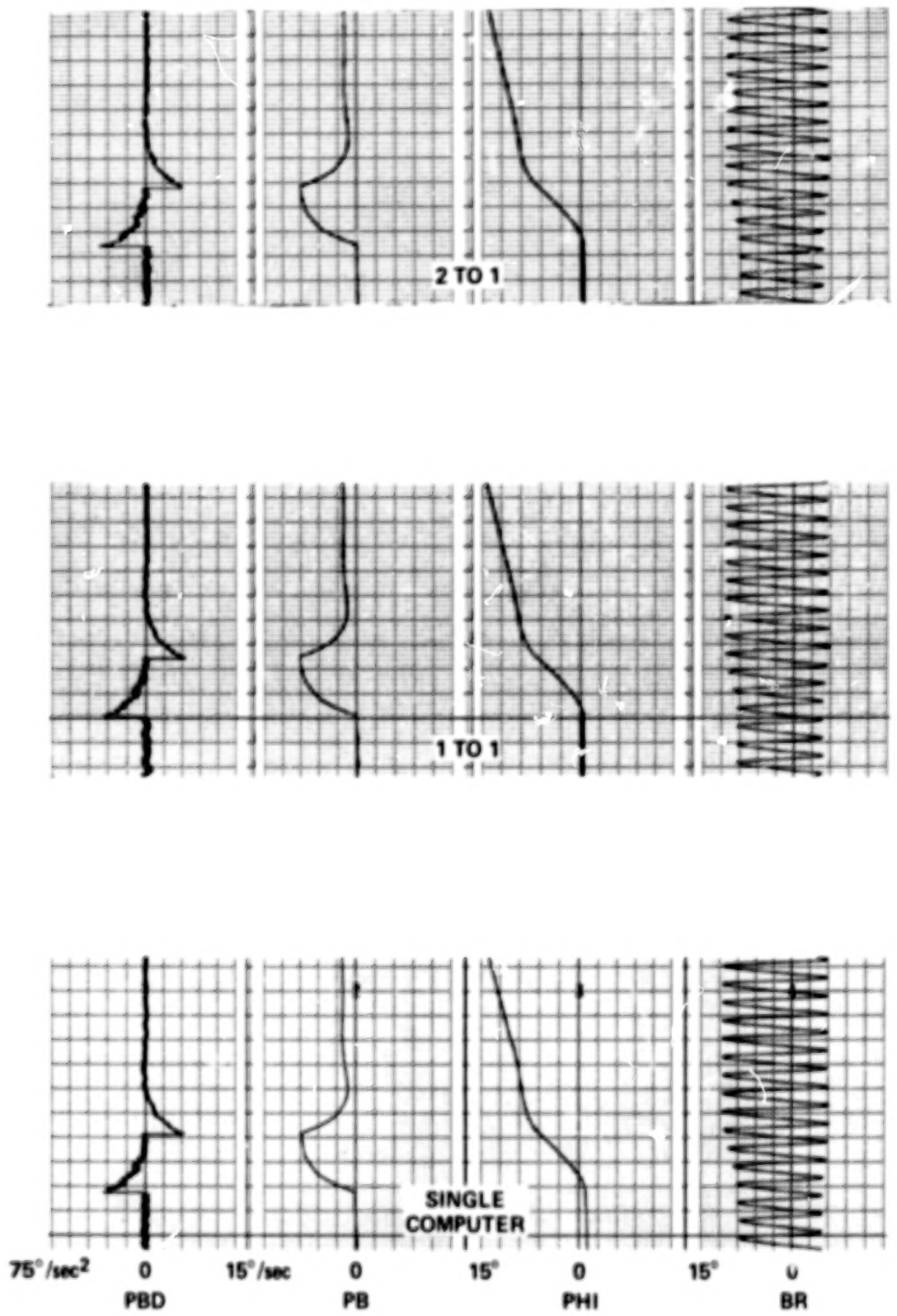


Figure 1.– Effect of increasing loop ratios on dynamic response of a compound rotorcraft at 220 knots. Rotor Δt = 5 msec. Computation of rotational rates and accelerations on host computer.

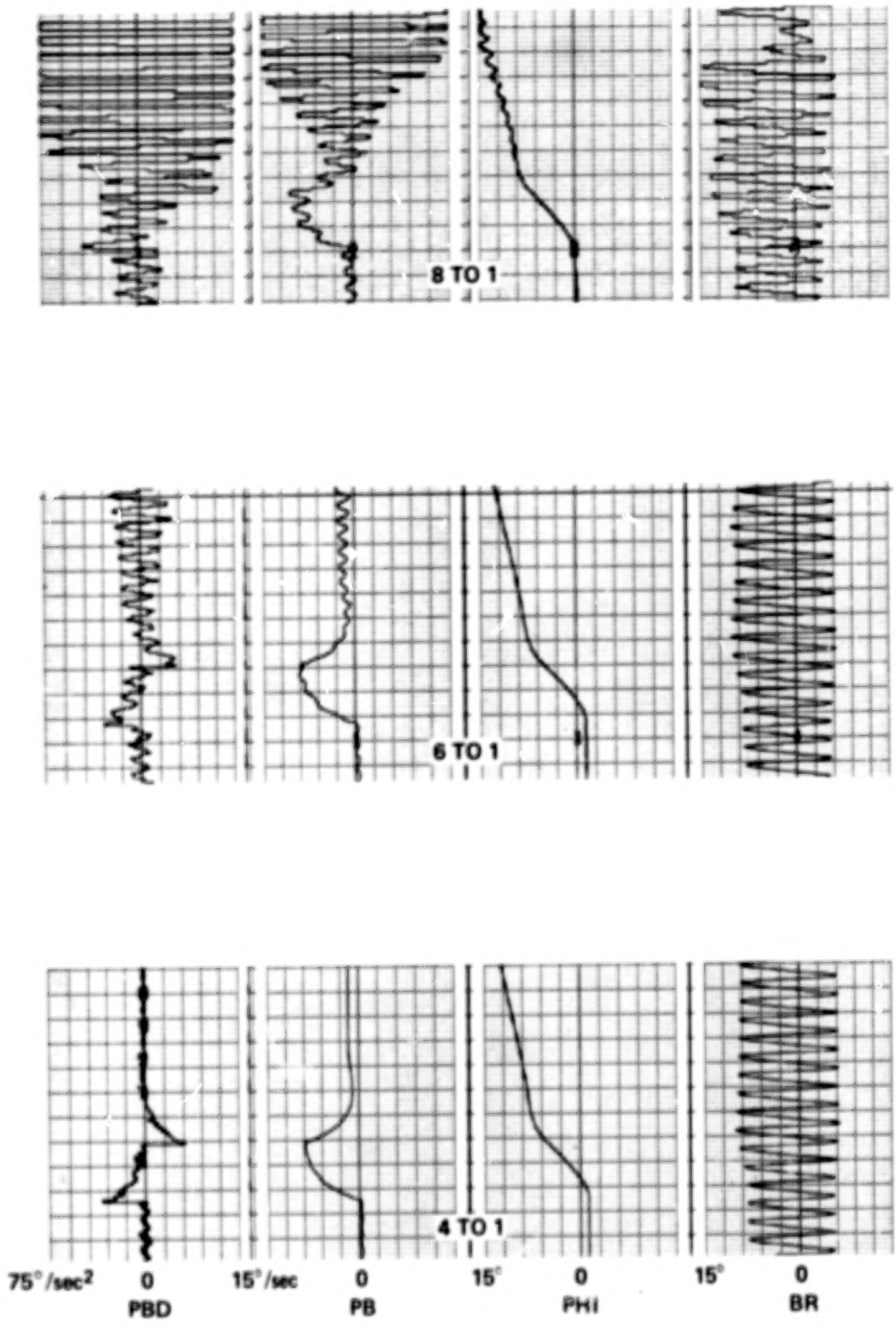


Figure 1.– Concluded.

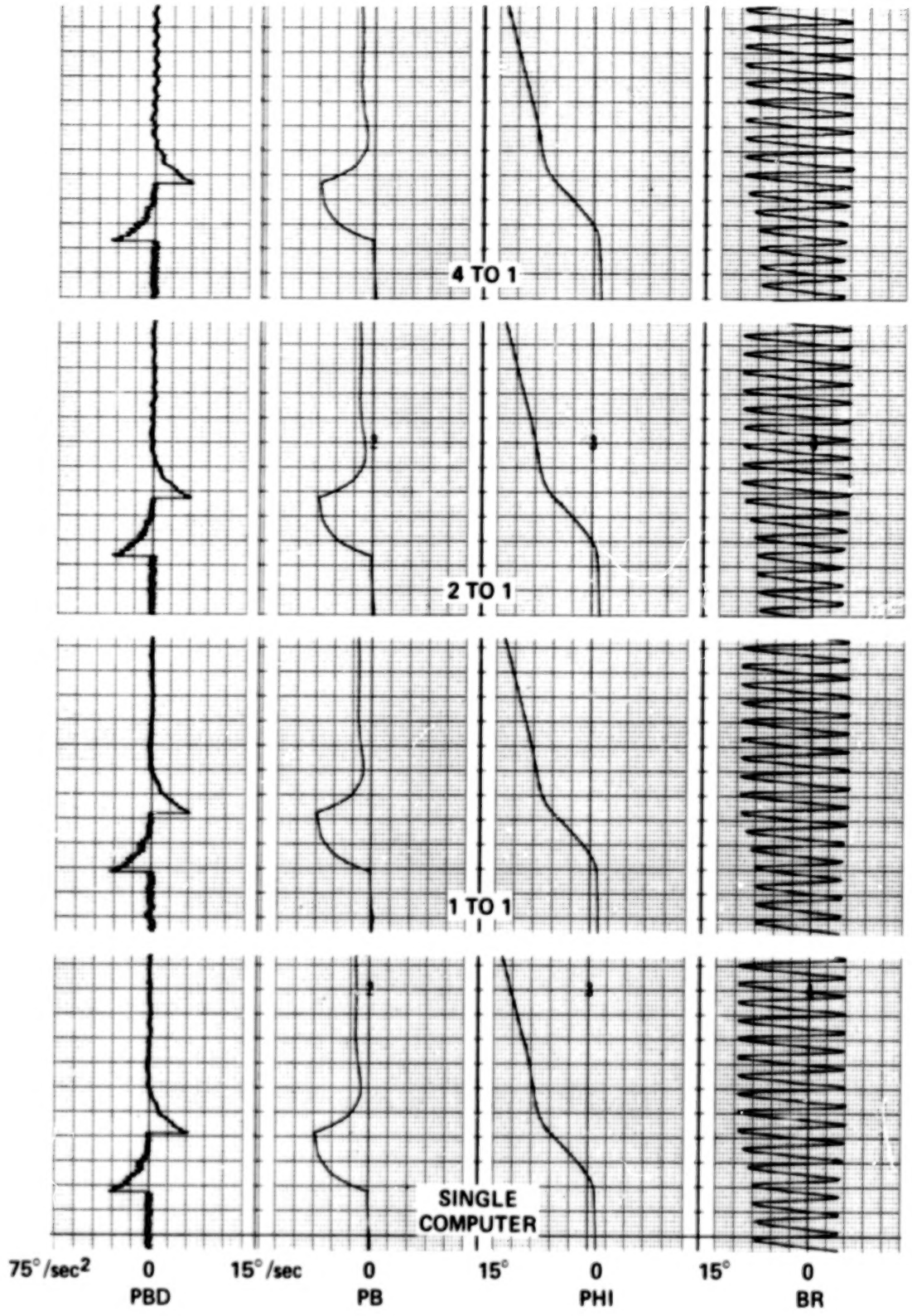


Figure 2.– Effect of increasing loop ratios on dynamic response of a compound rotorcraft at 220 knots. Rotor Δt = 5 msec. Computation of rotational rates and accelerations on array processor.

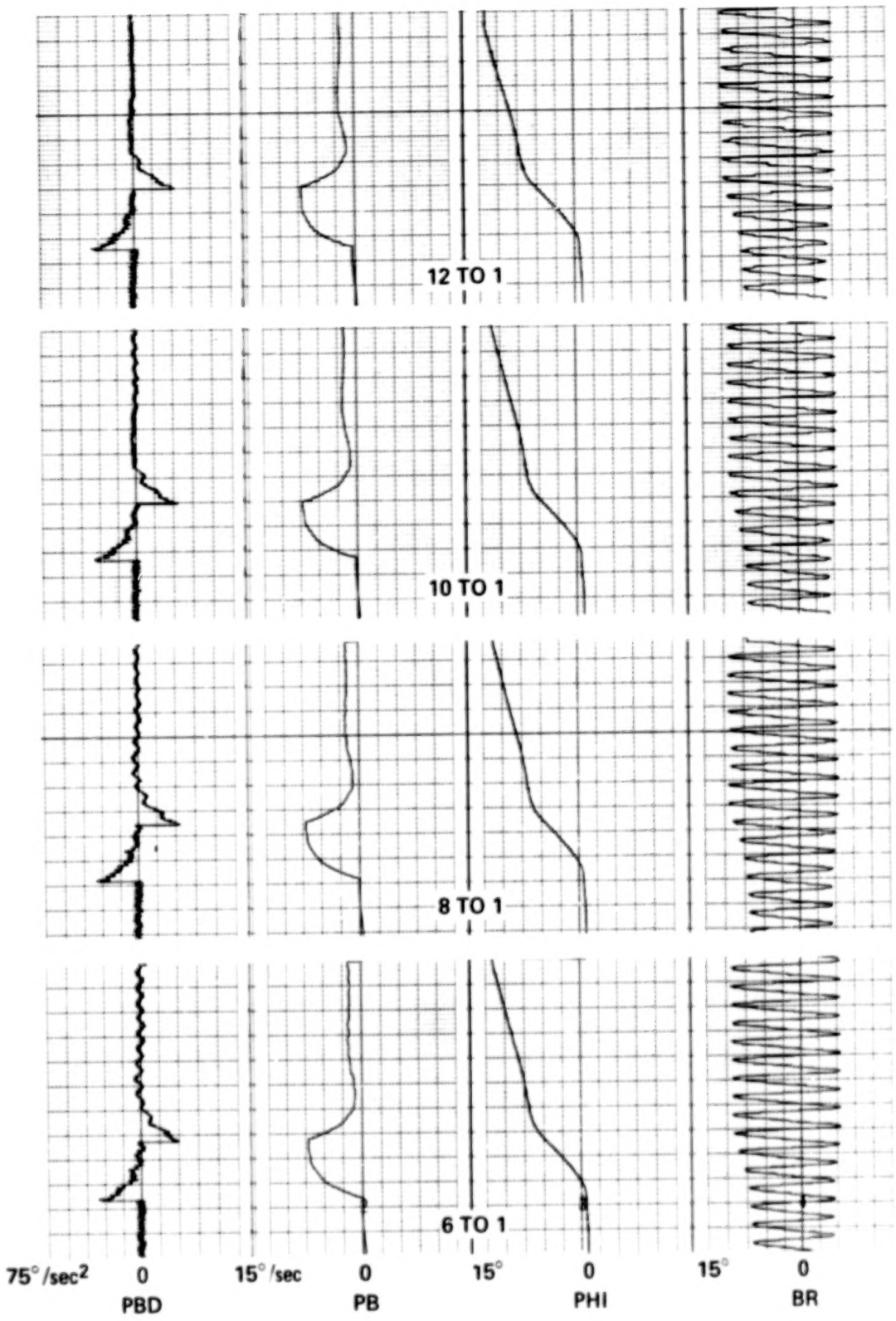


Figure 2.– Concluded.

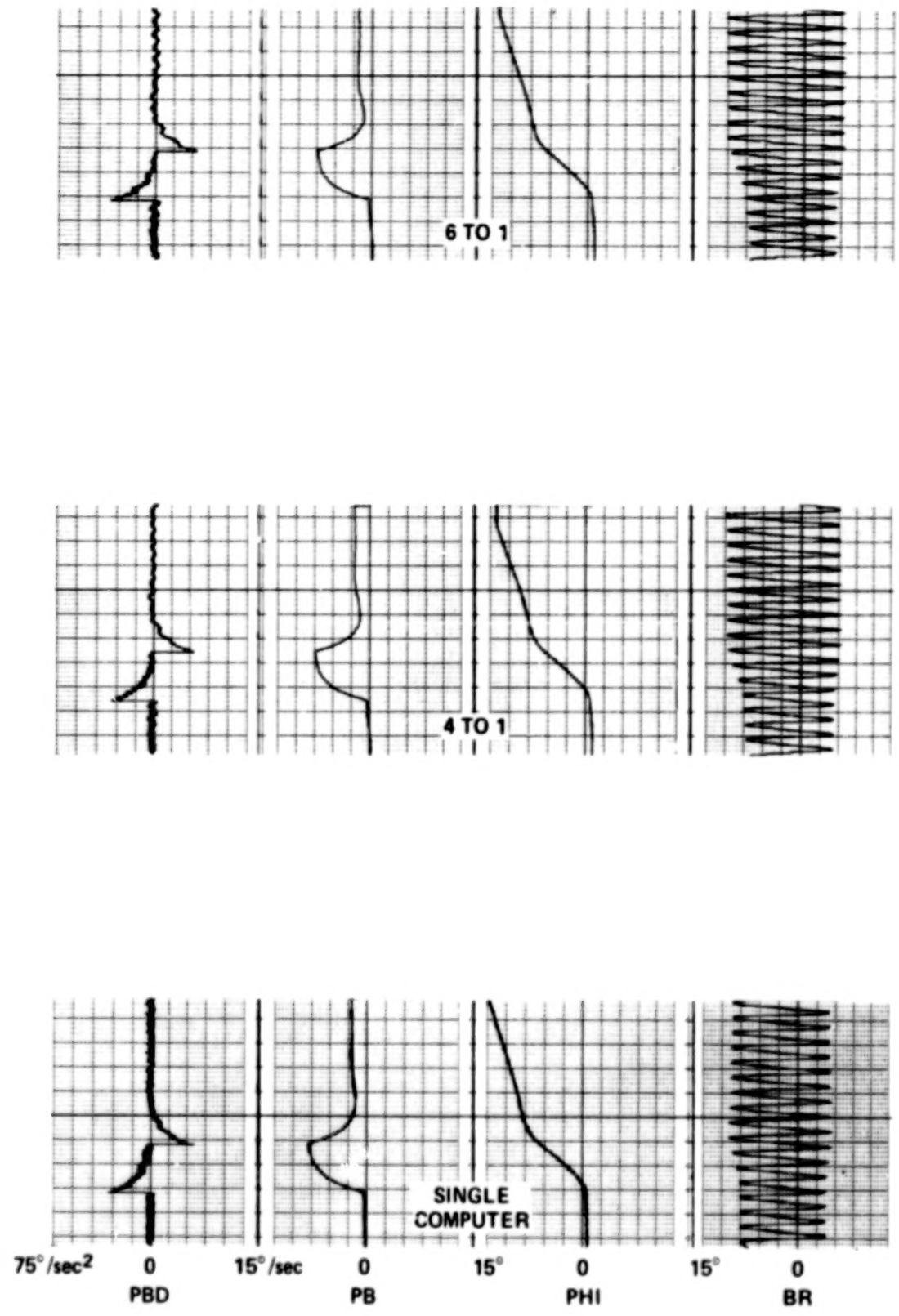


Figure 3.– Effect of increasing loop ratios on dynamic response of a compound rotorcraft at 220 knots. Rotor Δt = 6 msec. Computation of rotational rates and accelerations on array processor.

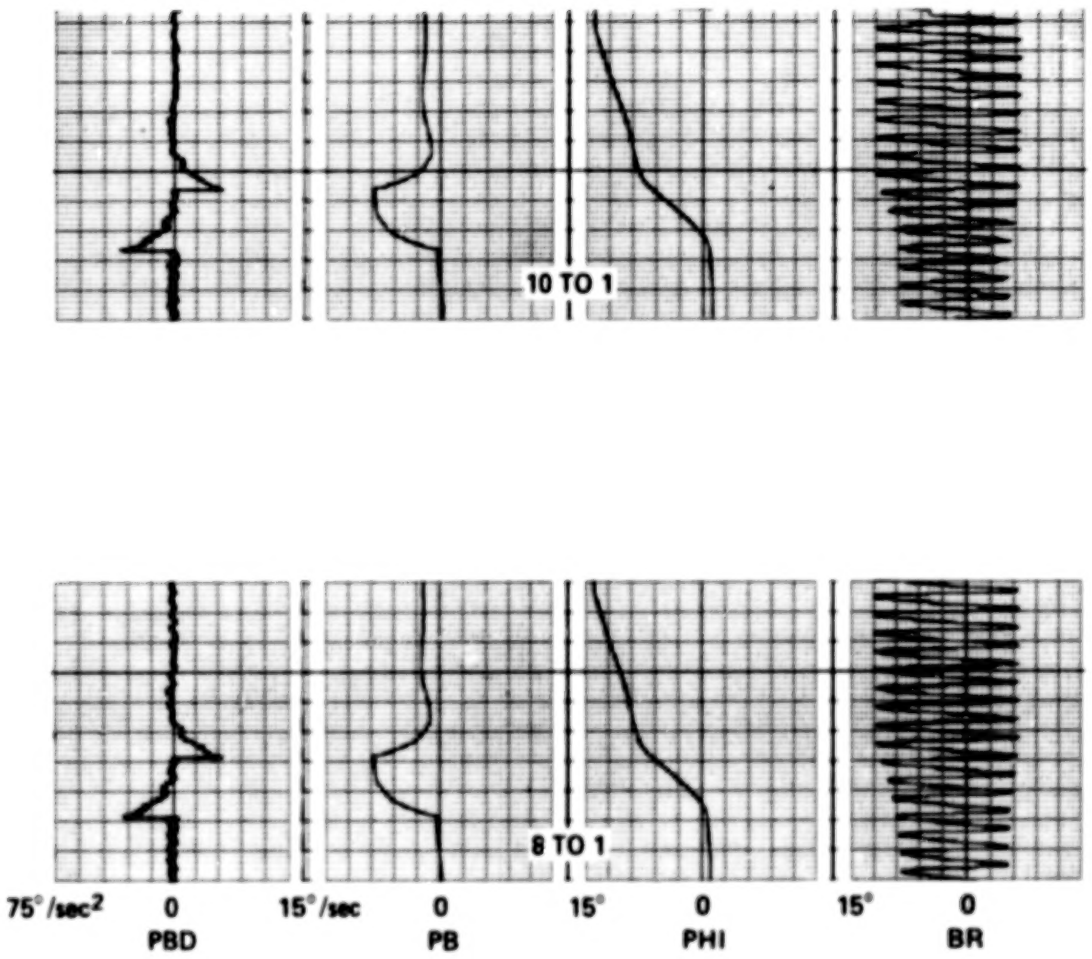


Figure 3.– Concluded.

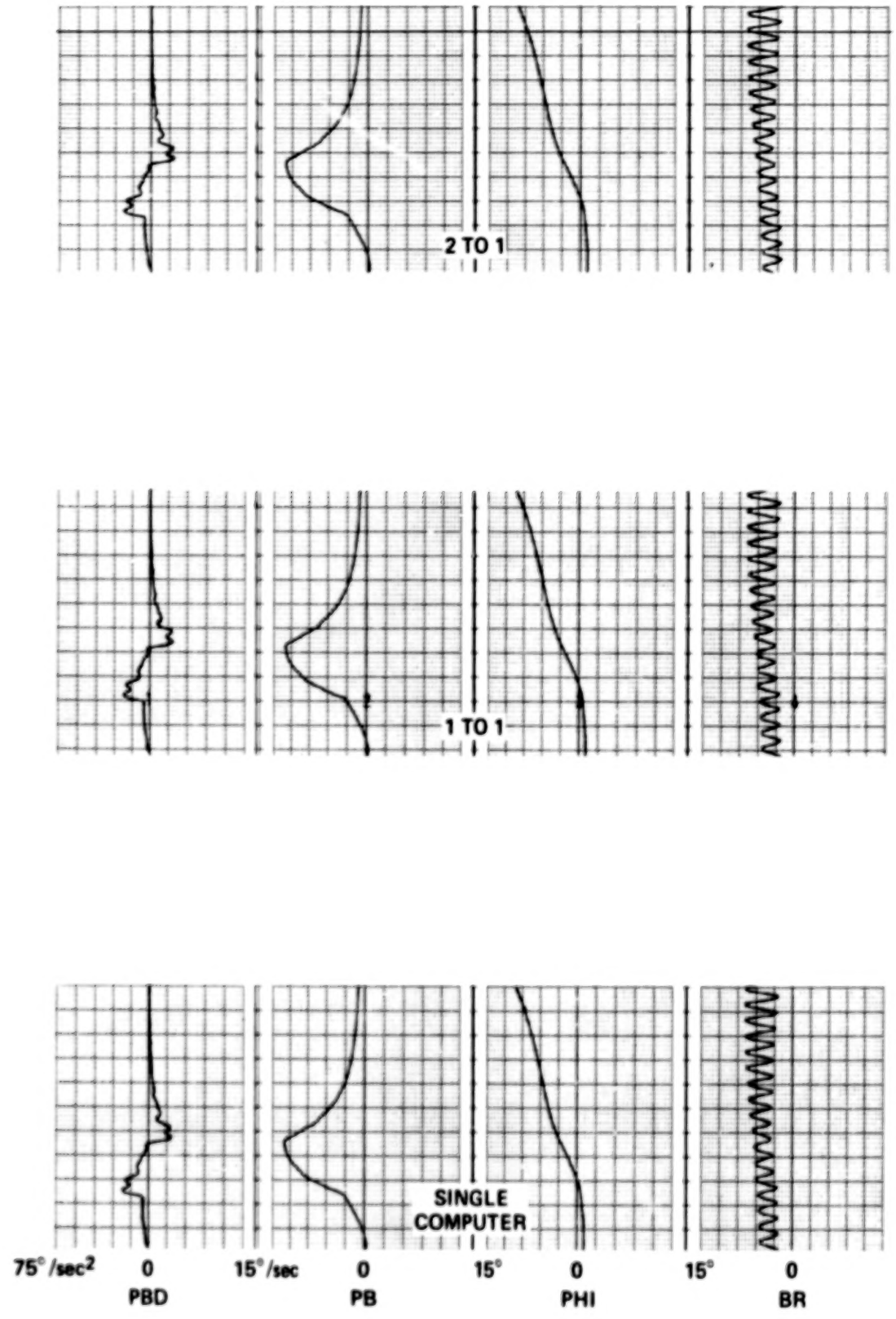


Figure 4.– Effect of increasing loop ratios on dynamic response of a helicopter at 60 knots. Rotor Δt = 5 msec. Computation of rotational rates and accelerations on array processor.

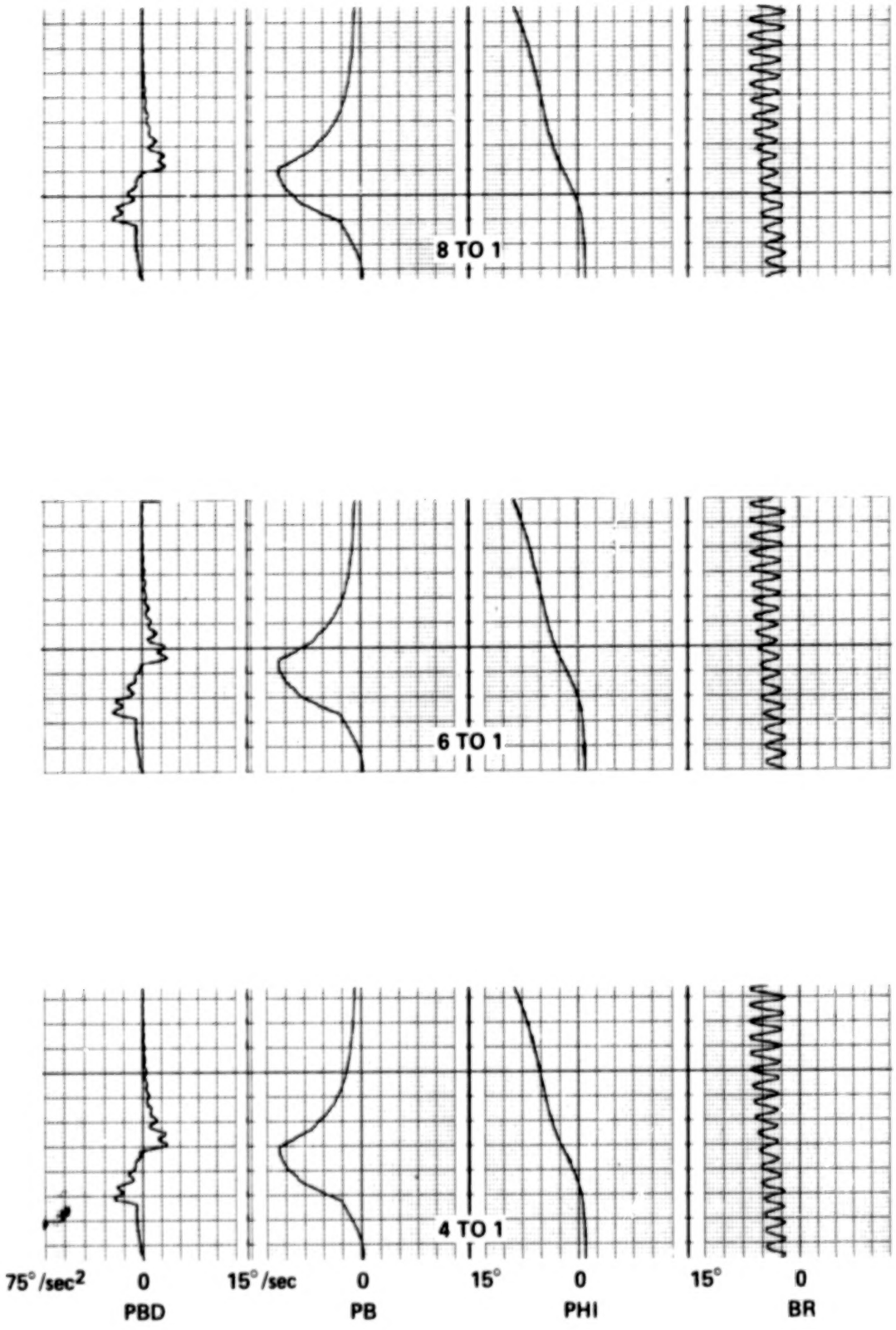


Figure 4.– Concluded.

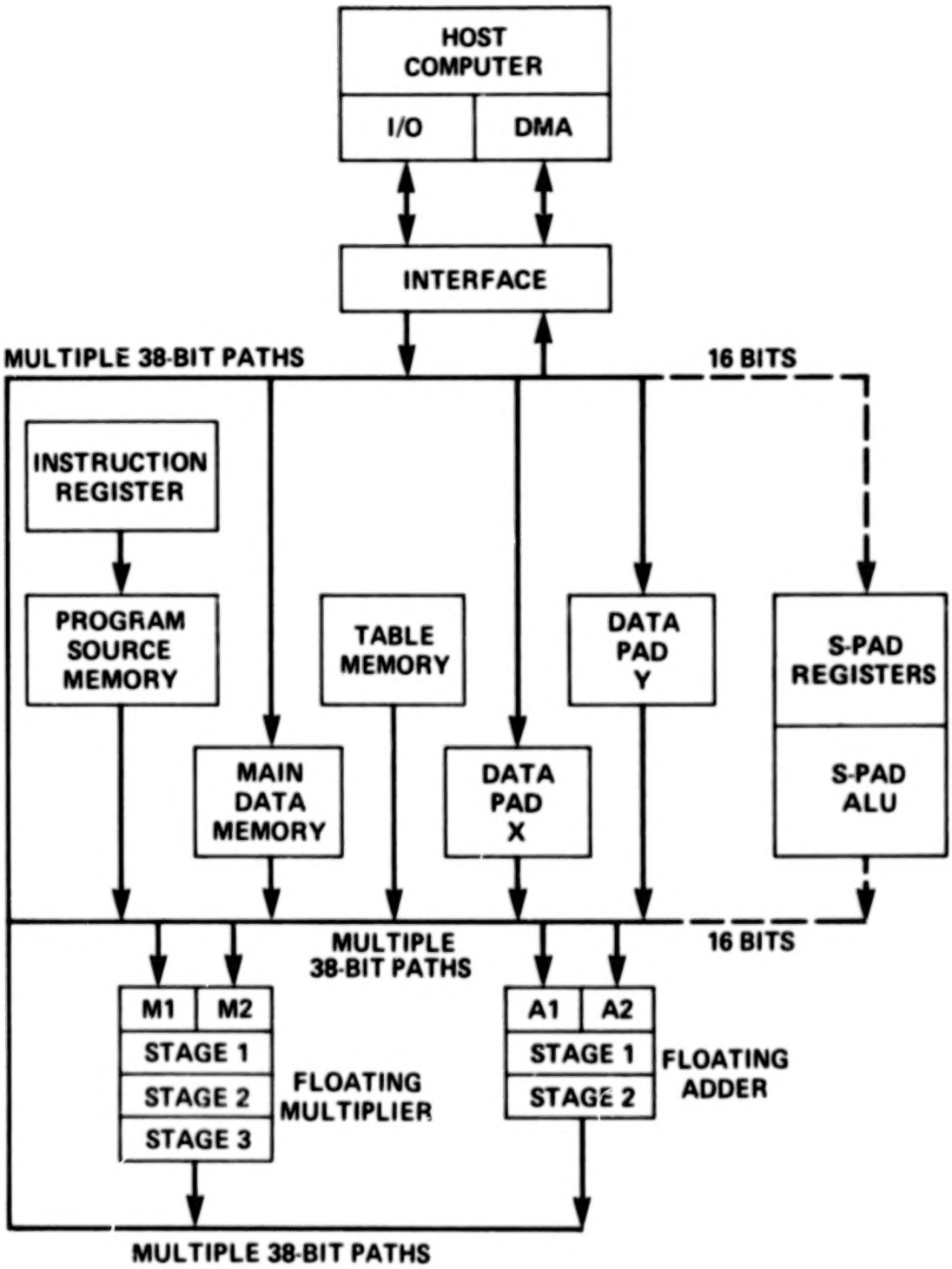


Figure 5.– Representative host/AP-120B block diagram.

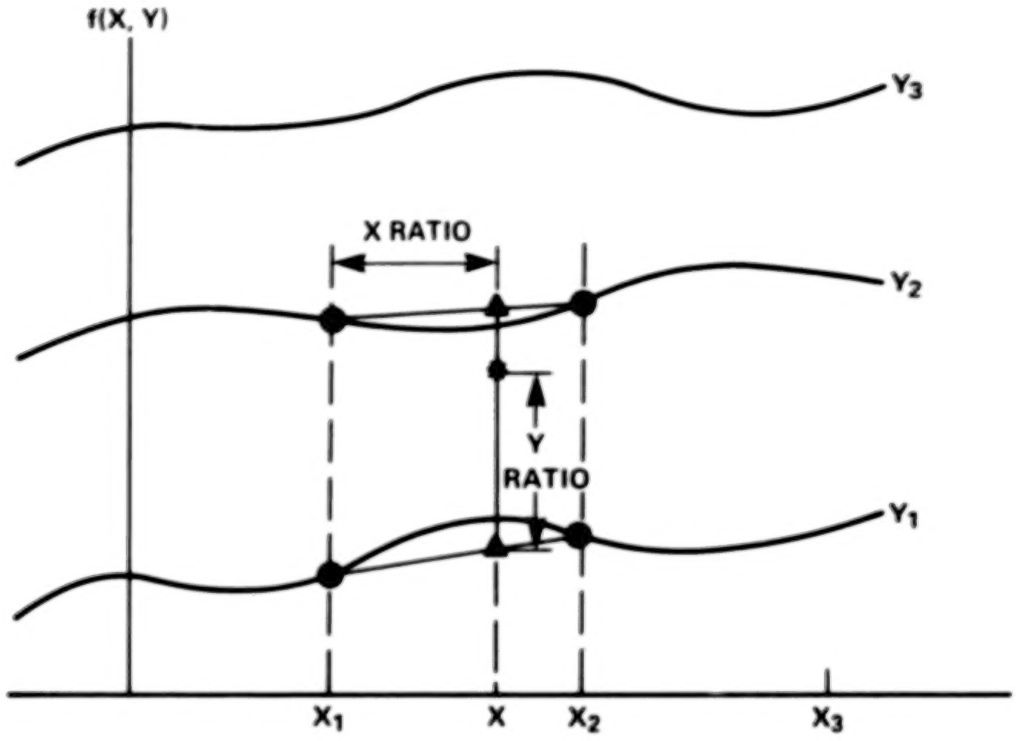


Figure 6.– Graphical representation of the table look-up procedure for a function of two variables.

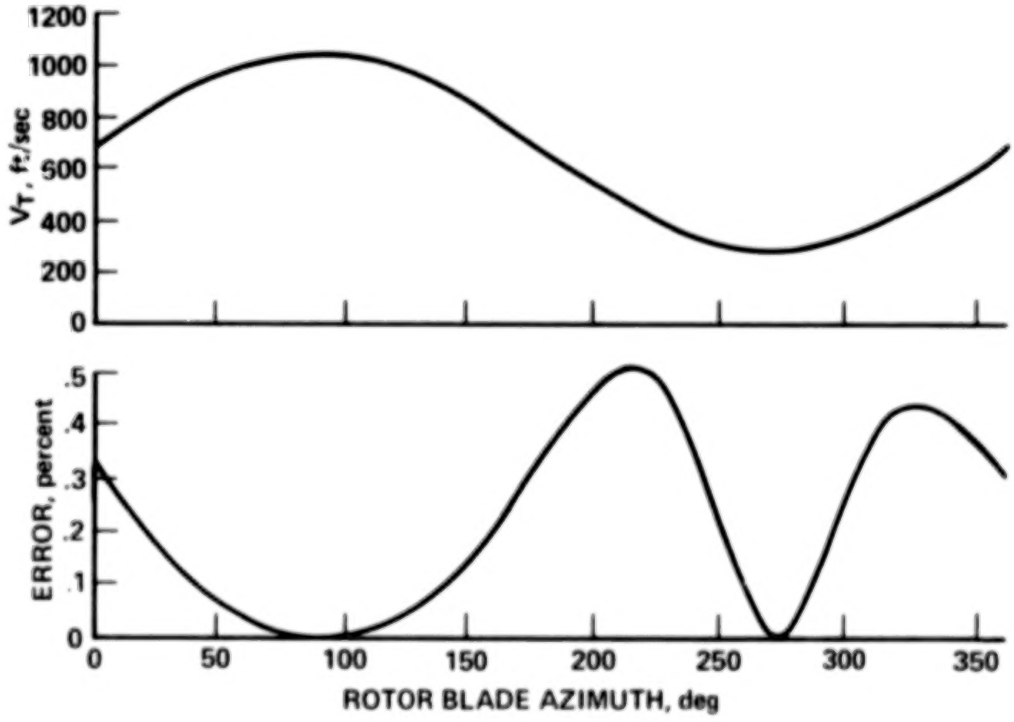


Figure 7.– Error in computation of single iteration inverse of V_T for 113 m/sec (220 knots) case V_T shown for reference.

1 Report No. NASA TP - 1267	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle APPLICATION OF SPECIAL-PURPOSE DIGITAL COMPUTERS TO ROTORCRAFT REAL-TIME SIMULATION		5. Report Date July 1978 6. Performing Organization Code
7. Author(s) D. Brian Mackie and Seth Michelson		8. Performing Organization Report No. A-7343 10. Work Unit No. 505-09-42
9. Performing Organization Name and Address NASA Ames Research Center, Moffett Field, Calif. 94035 and Computer Sciences Corporation, Mountain View, Calif. 94040		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Technical Paper 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

A study was initiated to determine the suitability of using an array processor as a computational element in rotorcraft real-time simulation. It was necessary to determine whether the speed of such a processor would be great enough to accurately simulate complex rotorcraft. Since memory limits on such a computer are quite restrictive, only the rotor portion of the model could be array processor resident. The array processor proved fast enough to execute the rotor code in less than 5 msec. Thus, the course of investigation branched into a study of the validity of a multilooping scheme, in which the rotor would loop over its calculations a number of times while the remainder of the model cycled once on a host computer. To prove that such a method would realistically simulate rotorcraft, a FORTRAN program was constructed to emulate a typical host/array processor computing configuration. The multilooping of an expanded rotor model, which included appropriate kinematic equations, resulted in an accurate and stable simulation.

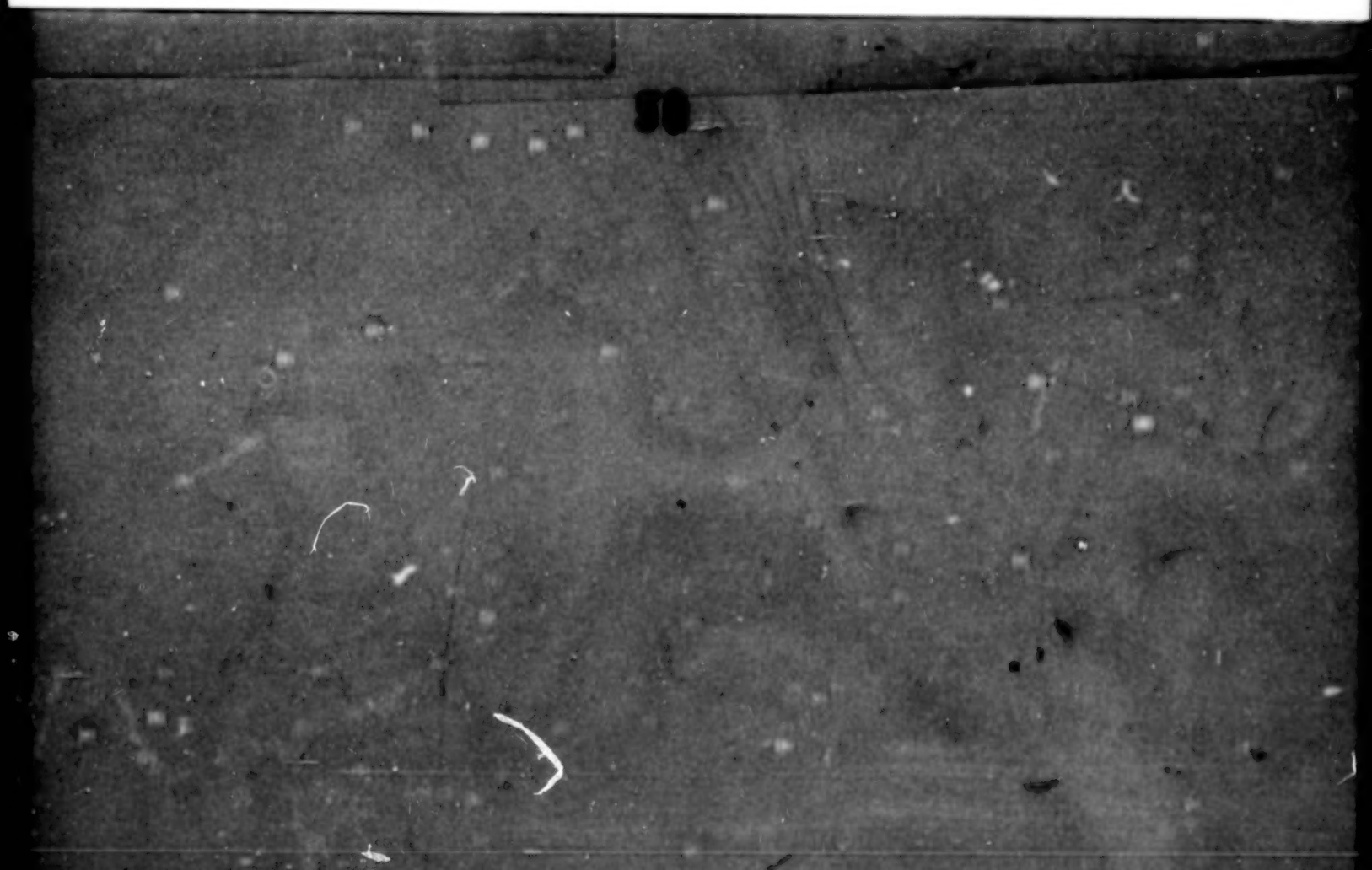
In the course of the study, many programming and operational difficulties were encountered. All of these were due to a fundamental conflict of concepts between the general-purpose program and the special-purpose computer. Some of these problems were mere inconveniences. However, several severe problems were also encountered during the development of the array processor program. In particular, major manpower effort was required for translation of a FORTRAN model into microcode, and logical debug would further require substantial effort.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Array processor Rotorcraft real-time simulation Multilooping	Unlimited STAR Category 05

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	35	$4.00

*For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1978



END

DEC 8 1978